Estimating the Mean and Variance of the Target Probability Distribution

David A. Nix and Andreas S. Weigend
Department of Computer Science and Institute of Cognitive Science
University of Colorado at Boulder
Boulder, CO 80309-0430, USA
dnix@cs.colorado.edu

***Abstract*— We introduce a method that estimates the mean and the variance of the probability distribution of the target as a function of the input, given an assumed target error-distribution model. Through the activation of an auxiliary output unit, this method provides a measure of the uncertainty of the usual network output for each input pattern. We here derive the cost function and weight-update equations for the example of a Gaussian target error distribution, and we demonstrate the feasibility of the network on a synthetic problem where the true input-dependent noise level is known.**

I. Introduction

Feed-forward artificial neural networks are widely used and well-suited for function-approximation (regression) tasks, particularly when there exists a sufficiently large data set from which to train. In almost any real-world problem, paired input-target data contains noise, one form of which is observational noise that corrupts the target values (e.g., [1]). Thus, when we attempt to approximate the function $f(\vec{x})$, we assume our measured data $d(\vec{x})$ can be modeled by

$$d(\vec{x}) = f(\vec{x}) + n(\vec{x}) \tag{1}$$

where the additive noise $n(\vec{x})$ can be viewed as errors on the target values that move the targets away from their true values $f(\vec{x})$ to their observed values $d(\vec{x})$. In a function-approximation task, the output of a network given a particular input pattern, $y(\vec{x})$, can be interpreted as an estimate $\hat{\mu}(\vec{x})$ of the true mean $\mu(\vec{x})$ of this noisy target distribution around $f(\vec{x})$, given an appropriate error model for $n(\vec{x})$ [2] [3] [4].

While an estimate of the mean of the target distribution (the expectation value) for a given input pattern is indeed valuable information, we sometimes want to know more. In addition to the network's predicting $d(\vec{x})$ by estimating the mean of the target distribution ($y(\vec{x}) = \hat{\mu}(\vec{x})$), we would also like the network to quantify the uncertainty of its prediction by simultaneously estimating the degree of noise about $\hat{\mu}(\vec{x})$ based on the noise observed in the training data (see [2] [5]).

Just as the network output y varies with the input pattern $\vec{x}$, the quantitative uncertainty due to $n(\vec{x})$, defined as the true variance σ^2 of the target error distribution around $f(\vec{x})$, is also some function of $\vec{x}$. This function, $\sigma^2(\vec{x})$, may be constant (i.e., independent of the input) if the target noise level is uniform over the range of input values. Alternatively–the case we want to consider here–the level of noise may vary systematically over the input space. In either case, not only do we want the network to learn an output function $y(\vec{x}) \approx f(\vec{x})$ that estimates the true mean $\mu(\vec{x})$ of the corresponding target distribution, but we also want to simultaneously learn a function $s^2(\vec{x})$ that estimates the true variance $\sigma^2(\vec{x})$ of that distribution, given an appropriate assumption as to the distribution's form.

Based on a maximum-likelihood formulation of a feed-forward neural network for function approximation [2] [3] [4], we here introduce a network that calculates $s^2(\vec{x}) \approx \sigma^2(\vec{x})$, the estimated variance of the target error distribution as a function of the input, in addition to the usual output $y(\vec{x}) = \hat{\mu}(\vec{x}) \approx \mu(\vec{x}) = f(\vec{x})$. We derive the method in full for the case where we assume the targets are normally distributed about $f(\vec{x})$, and we apply this derivation to a synthetic example problem where $\sigma^2(\vec{x})$ is known.

II. The Method

A. The Idea

How does the network estimate $s^2(\vec{x})$? To the output unit y that computes $\hat{\mu}(\vec{x}_i)$, we add a complementary "s^2 unit" that computes $s^2(\vec{x}_i)$, the estimate of $\sigma^2(\vec{x}_i)$ given input pattern $\vec{x}_i$.

Since $\sigma^2(\vec{x})$ can never be negative or zero, we choose an exponential activation function for $s^2(\vec{x})$ to naturally impose these bounds:

$$s^2(\vec{x}_i) = \exp\left[\sum_k w_{s^2k} h_k^{s^2}(\vec{x}_i) + \beta\right] \tag{2}$$

where β is the bias for the s^2 unit and $h_k^{s^2}(\vec{x}_i)$ is the activation of hidden unit k for input $\vec{x}_i$ in the hidden layer feeding directly into the s^2 unit.

Having selected a particular network architecture (see Figure 1; inputs $\vec{x}$ are indexed by m, hidden units by j and k), we employ the same gradient-descent (backpropagation) learning as in the usual case to find w_{jm} and w_{yj} in order to also find a set of weights w_{km} and w_{s^2k} that calculate $s^2(\vec{x})$. Thus, after each pattern i is presented, all the weights in the network[1] are adapted to minimize some cost function C according to

$$\Delta w_{yj} = -\eta \frac{\partial C_i}{\partial w_{yj}} \tag{3}$$

$$\Delta w_{jm} = -\eta \frac{\partial C_i}{\partial w_{jm}} \tag{4}$$

$$\Delta w_{s^2k} = -\eta \frac{\partial C_i}{\partial w_{s^2k}} \tag{5}$$

$$\Delta w_{km} = -\eta \frac{\partial C_i}{\partial w_{km}} \tag{6}$$

where η is the learning rate and C_i is the contribution of pattern i to the overall cost function C.

We obtain a form for C by expressing our goal as maximizing the log likelihood of the targets (having assumed our patterns are independently and identically distributed), given the input patterns and the network $\mathcal{N}$ (e.g., [2] [4] [6]). That is, we attempt to maximize

$$\sum_i \ln P(d_i|\vec{x}_i, \mathcal{N}). \tag{7}$$

The exact form of C depends on the assumption we make as to the form of this target probability distribution such that $y(\vec{x}) = \hat{\mu}(\vec{x})$.

B. Details

B.1. Architecture

The s^2 unit is fully connected to its own set of hidden units, h^{s^2} (indexed by k), just as the output unit y is connected to its hidden units h^y (indexed by j) (see Figure 1). Alternatively, we could connect both y and s^2 to a common large set of hidden units, but our experience has been that the method works better when we use a split-hidden-unit architecture. In addition, we could easily add a second hidden layer (split or shared) as required by the particular form of $f(\vec{x})$ and/or $\sigma^2(\vec{x})$, but we will restrict our attention here to the case of a single hidden layer.

B.2. Learning Dynamics

All initial weights are drawn from a uniform random distribution $\in [-1, 1]$ and scaled by the reciprocal of the number of incoming connections. These

[1]The biases of all units are considered to be an additional weight connected to a hidden unit clamped at 1 and are updated accordingly.

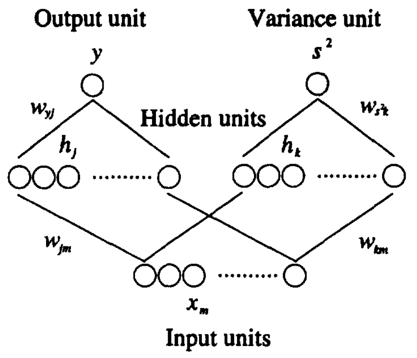

Figure 1: Architecture for a network with one output unit y and one s^2 unit. Both sets of hidden units are connected to the same input units, but no connections are shared (not all connections are shown).

initial weights produce a net input to the s^2 unit's exponential activation function (Eq. (2)) of approximately zero, corresponding to an initial estimate of approximately unit variance over the entire input space. Because it would be premature to make differing estimates of the noise level over the input space before $f(\vec{x})$ is at least roughly approximated by $y(\vec{x})$, we save computations by not updating the weights w_{km} and w_{s^2k} until $y(\vec{x})$ is somewhat close to $f(\vec{x})$.

Additionally, if the variance is either much larger or much smaller than unity, the bias β in Eq. (2), will have to grow either very positive of very negative for $s^2(\vec{x})$ to well approximate $\sigma^2(\vec{x})$. Since β has a natural interpretation as the natural log of the mean value of $\sigma^2(\vec{x})$, we can accelerate the learning of $s^2(\vec{x})$ by setting β equal to the natural log of the current mean variance over the training data at the endof each epoch (for early training epochs). The precise form of the variance depends on the assumed form of the target error distribution model (see example below).

However, once training has proceeded to the point where the approximation of $f(\vec{x})$ is improving only very slowly, we assume that $y(\vec{x})$ is a rough approximation of $f(\vec{x})$ that additional training will fine-tune. At this point, and for subsequent training, we update the weights that calculate $s^2(\vec{x})$ according to Eqs. (13) and (14). Furthermore, we no longer set the s^2 unit's bias β to the natural log of the current mean variance over the training data; instead we update β according to gradient descent just like all other weights and biases. Training is then continued until C does not decrease significantly further.

III. A Specific Example

A. *Normally Distributed Errors*

Least-squares regression techniques can be interpreted as maximum likelihood with an underlying Gaussian error model. In this simple case of assuming normally distributed errors around $f(\vec{x})$, we have

$$P(d_i|\vec{x}_i,\mathcal{N}) = \frac{1}{\sqrt{2\pi\sigma^2(\vec{x}_i)}} \exp\left(\frac{-[d_i - y(\vec{x}_i)]^2}{2\sigma^2(\vec{x}_i)}\right) \tag{8}$$

as the target probability distribution for input pattern $\vec{x}_i$, where, as before, $y(\vec{x}_i)$ corresponds to the mean of this distribution and $\sigma^2(\vec{x}_i)$ is the variance.

If we take the natural log of both sides, we get

$$\begin{aligned}\ln P(d_i|\vec{x}_i,\mathcal{N}) &= -\frac{1}{2}\ln(2\pi) \\ &\quad -\frac{1}{2}\ln[\sigma^2(\vec{x}_i)] - \frac{[d_i - y(\vec{x}_i)]^2}{2\sigma^2(\vec{x}_i)} \quad (9)\end{aligned}$$

as the log likelihood to be maximized. The first term on the right is a constant and can be ignored for maximization. Since maximizing a value is the same as minimizing the negative of that value, we write what remains of the right-hand side of Eq. (9) as a cost function C to be minimized over all patterns i:

$$C = \sum_i \frac{1}{2}\left(\frac{[d_i - y(\vec{x}_i)]^2}{\sigma^2(\vec{x}_i)} + \ln[\sigma^2(\vec{x}_i)]\right). \tag{10}$$

Using Eq. (10) for C, we obtain our weight-update equations by first specifying a linear activation function for y and tanh activation functions for the hidden units.[2] Then we approximate $\sigma^2(\vec{x}_i)$ by $s^2(\vec{x}_i)$, the activation of the s^2 unit, and take the derivatives in Eqs. (3)–(6) for pattern i:

$$\begin{aligned}\Delta w_{yj} &= \eta\,\frac{1}{s^2(\vec{x}_i)}[d_i - y(\vec{x}_i)]h_j^y(\vec{x}_i) \quad (11)\\ \Delta w_{jm} &= \eta\,\frac{1}{s^2(\vec{x}_i)}[d_i - y(\vec{x}_i)] \\ &\quad \times\ w_{yj}[1 - h_j^y(\vec{x}_i)]^2 x_{m,i} \quad (12)\\ \Delta w_{s^2k} &= \eta\,\frac{1}{2}\left(\frac{[d_i - y(\vec{x}_i)]^2}{s^2(\vec{x}_i)} - 1\right)h_k^{s^2}(\vec{x}_i) \quad (13)\\ \Delta w_{km} &= \eta\,\frac{1}{2}\left(\frac{[d_i - y(\vec{x}_i)]^2}{s^2(\vec{x}_i)} - 1\right) \\ &\quad \times\ w_{s^2k}[1 - h_k^{s^2}(\vec{x}_i)]^2 x_{m,i}. \quad (14)\end{aligned}$$

Despite one of the tasks of the *Santa Fe Time Series Prediction and Analysis Competition* having been to generate error bars in addition to the predicted values themselves, none of the entries contained principled uncertainty estimates [7]. Ignoring the possibility a variable $\sigma^2(\vec{x})$ is equivalent to assuming $\sigma^2(\vec{x}_i)$ to be a constant independent of $\vec{x}_i$. With this assumption, the second term in Eq. (10) is a constant that can be ignored for minimization, and the $1/2\sigma^2(\vec{x}_i)$ term in Eq. (10) is a constant that is incorporated into the learning rate η in Eqs. (11) and (12). This assumption results in the standard equations for backpropagation using a sum-squared-error cost function. However, since we are specifically allowing for a variable $\sigma^2(\vec{x}_i)$, we explicitly keep these terms in the cost function.

[2] We could select any appropriate hidden-unit activation function for either set of hidden units; however, for simplicity we choose all tanh activation functions.

B. *A Synthetic Example Problem*

B.1. *The Problem*

To demonstrate the application of Eqs. (10)–(14), we construct a one-dimensional example problem where the true $f(\vec{x})$ and $\sigma^2(\vec{x})$ are known. We consider an amplitude-modulation equation of the form

$$f(x) = m(x)\sin(\omega_c x) \tag{15}$$

where $m(x) = \sin(\omega_m x)$. For this simple example we choose $\omega_c = 5$ and $\omega_m = 4$ over the interval $x \in [0, \pi/2]$.

We generate our target values according to Eq. (1) where $n(x)$ is zero-mean Gaussian noise with variance $\sigma^2(x)$ that changes according to

$$\sigma^2(x) = 0.02 + 0.02 \times [1 - m(x)]^2. \tag{16}$$

We generate 5000 patterns and randomly assign approximately 25% of these to a cross-validation set to guard against overfitting [8]. We perform gradient-descent learning on the remainder of the patterns, updating the weights according to Eqs. (11)–(14) after the presentation of each pattern. We connect thirty tanh hidden units to y and ten tanh hidden units to s^2. To avoid artifacts, we use a conservative learning rate of $\eta = 10^{-5}$ and do not use momentum.

As described above, initially only the weights and biases that calculate $y(x)$ are updated after each pattern presentation, and β in Eq. (2) is set at the end of each epoch to the natural log of the mean-squared error over the entire training set. After the normalized mean-squared error stops decreasing sharply, *all* parameters in the network are adapted by Eqs. (11)–(14) after each pattern presentation, including β.

B.2. *Results*

The learning curve for the normalized-mean-squared error (NMSE) is plotted in Figure 2. The curve

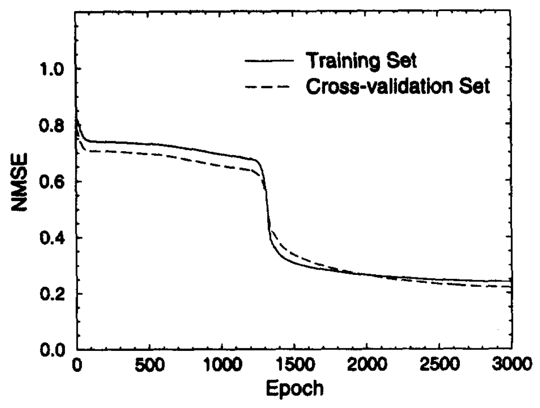

Figure 2: Learning curve for normalized mean-squared error ($NMSE_{\mathcal{D}} = MSE_{\mathcal{D}}/\sigma^2_{\mathcal{D}}$; where $\mathcal{D}$ is either the training or the cross-validation data).

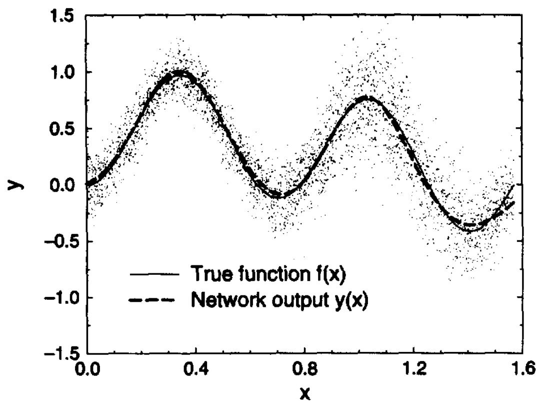

Figure 4: Training data, true function $f(x)$, and estimate $y(x)$ (epoch 3000).

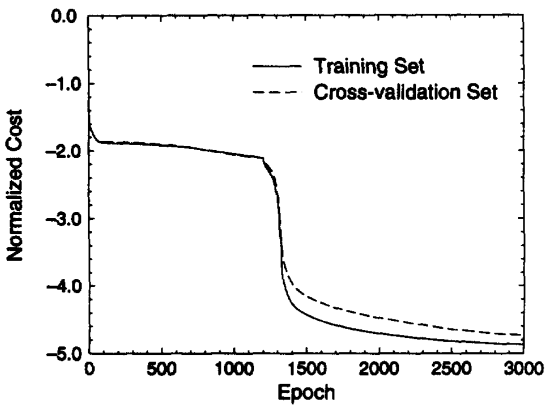

Figure 3: Learning curve for normalized cost ($NC_{\mathcal{D}} = C_{\mathcal{D}}/\sigma^2_{\mathcal{D}}$; where $\mathcal{D}$ is either the training or the cross-validation data).

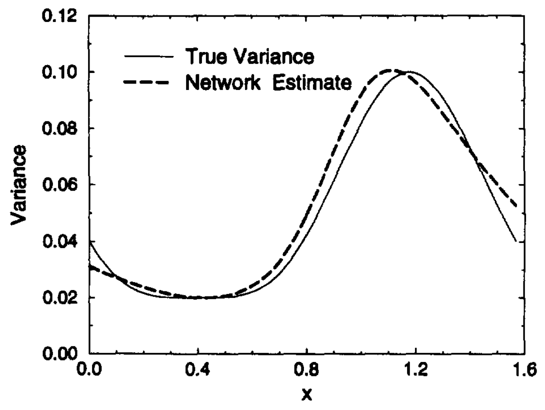

Figure 5: True variance $\sigma^2(x)$ and estimate $s^2(x)$ (epoch 3000).

has two primary descent phases, corresponding to learning each of the two significant oscillations in $f(x); x \in [0, \pi/2]$. The curve starts to level out at about epoch 1500, at which point $y(x)$ approximates the gross features of $f(x)$. After epoch 1500, Eqs. (11)-(14) are used to update *all* parameters in the network.

While the NMSE continues to decrease slightly as the approximation $y(x) \approx f(x)$ is fine-tuned, we see in Figure 3 that the normalized cost (NC) continues to decrease steadily as $s^2(x)$ learns to approximate $\sigma^2(x)$. Training is continued until epoch 3000. Note that no overfitting with respect to either the NMSE or the NC is observed on the cross-validation set.

In Figure 4 we plot the training data, the true function $f(x)$, and the output of the network, $y(x)$, over the interval $x \in [0, \pi/2]$. We see that the network's approximation $y(x)$ closely matches the shape of $f(x)$. Note that the approximation is slightly better for smaller x than for larger x; this is due in part to the robust regression effect described below.

The true variance $\sigma^2(x)$, given by Eq. (16), is plotted in Figure 5 along with the network's estimate $s^2(x)$ over the interval $x \in [0, \pi/2]$. We see that $s^2(x)$ follows closely the shape of $\sigma^2(x)$. The slight differences are due to a combination of two factors. First, the approximation of $f(x)$ is not perfect, so there is some error introduced to the approximation of $\sigma^2(x)$. Second, with a finite sample size the target noise will not have exactly the true $\sigma^2(x)$. With these slight differences aside, however, for a given input x, we have accurate estimates y and s^2 of both the mean and the variance of the target probability distribution.

IV. Discussion

A. *Robust Regression*

Naively, one might expect that allowing for variations in $\sigma^2(\vec{x})$ (with the addition of the s^2 unit and the resulting modifications of the standard backpropagation weight-update equations) does not alter the way the network approximates $f(\vec{x})$. However, this is *not* the case. According to Eqs. (11) and (12), as long as $\sigma^2(\vec{x})$ is constant over all $\vec{x}_i$, the effective learning rate is constant over all patterns (η/σ^2). However, for input patterns where $\sigma^2(\vec{x})$ is smaller than average, the learning rate η is effectively amplified compared to patterns for which $\sigma^2(\vec{x})$ is larger that average. Thus, this particular estimation of $\sigma^2(\vec{x})$ has the side-effect of biasing the network's allocation of its resources towards lower-noise regions, discounting regions of the input space where the network is producing larger-than-average errors. Through this side-effect, this procedure implements a form of robust regression, emphasizing low-noise regions of the input space in the allocation of the network's remaining resources.

B. *Overfitting*

In the example problem above, a sufficiently large data set was used such that overfitting of $y(\vec{x})$ to the training data was not observed. However, in most applications data sets are more limited, and overfitting can present a serious problem. In our method, an accurate approximation of $\sigma^2(\vec{x})$ depends on the quality of $y(\vec{x})$ as an approximation of $f(\vec{x})$ without overfitting. To see why this is so, consider the extreme case where a network overfits the training data such that the error is zero on every training pattern. Then the estimated variance would be zero even though in reality there may be considerable target noise about the true $f(\vec{x})$.

In addition, we must also be concerned with overfitting $s^2(\vec{x})$ to the training data. For example, take a situation in which the true variance is constant over the input space, yet in one small region we have four one-dimensional input patterns arranged such that the outer two patterns have small errors from $f(x)$ and and the inner two have large errors. We do not want $s^2(x)$ to estimate a sudden increase in the variance in the region of the inner two patterns. Therefore, in applying our technique to relatively short data sets, we must use the same anti-overfitting weaponry as is required when attempting function approximation with any sparse data set (e.g., adding complexity penalty terms to Eq. (10) [6] [8]).

V. Conclusions

We have introduced a method to estimate the uncertainty of the output of a network that tries to approximate a function. This is accomplished by learning a second function $s^2(\vec{x})$ that estimates $\sigma^2(\vec{x})$, the variance of the target probability distribution around $f(\vec{x})$ as a function of the input $\vec{x}$. This function provides a quantitative estimate of the target noise level depending on the location in input space and, therefore, provides a measure of the uncertainty of $y(\vec{x})$.

We have derived the specific weight update equations for the case of Gaussian noise on the outputs, i.e., we have shown how to estimate the second moment (variance) of the target distribution in addition to the usual estimation of the first moment (mean). The extension of this technique to other error models is straightforward (e.g., a Poisson model could be used when the errors are suspected to be Poisson distributed, etc.).

For very sparse data sets, we may only be able to reasonably estimate the first moment of the target distribution. Estimating both the first and second moments is a reasonable goal if we are dealing with a moderately sized data set. For extremely large data sets, however, one can be more ambitious and aim for estimating the entire probability density function using connectionist methods [9] or hidden Markov models with mixed states [10].

We will apply our method to the real-world Data Set A (from a laser) from the *Santa Fe Time Series Analysis and Prediction Competition* [5] [7].

Acknowledgments

We would like to thank David Rumelhart and Barak Pearlmutter for discussing the problem and the approach. We would also like to thank Wray Buntine for emphasizing the potential problem of overfitting of the variance function. This work was supported by a Graduate Fellowship from the Office of Naval Research and by NSF grant RIA ECS-9309786.

References

[1] M. Casdagli, S. Eubank, J.D. Farmer, and J. Gibson, "State Space Reconstruction in the Presence of Noise." *Physica D*, vol. 51D, pp. 52–98, 1991.

[2] W.L. Buntine and A.S. Weigend, "Bayesian Backpropagation." *Complex Systems*, vol. 5, pp. 603–643, 1991.

[3] D. MacKay, "A Practical Bayesian Framework for Backpropagation." *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[4] D.E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin, "Backpropagation: The Basic Theory." In *Backpropagation: Theory, Architectures and Applications,* Y. Chauvin and D. E. Rumelhart, eds., Lawrence Erlbaum, 1994.

[5] N.A. Gershenfeld and A.S. Weigend, "The Future of Time Series." In *Time Series Prediction: Forecasting the Future and Understanding the Past*, A.S. Weigend and N.A. Gershenfeld, eds., Addison-Wesley, pp. 1–70, 1994.

[6] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart, "Predicting Sunspots and Exchange Rates with Connectionist Networks." In *Nonlinear Modeling and Forecasting*, M. Casdagli and S. Eubank, eds., Addison-Wesley, pp. 395–432, 1992.

[7] A.S. Weigend and N.A. Gershenfeld, eds., *Time Series Prediction: Forecasting the Future and Understanding the Past.* Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. XV., Addison-Wesley, 1994.

[8] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart, "Predicting the Future: A Connectionist Approach," *International Journal of Neural Systems*, vol. 1, pp. 193–209, 1990.

[9] A.S. Weigend, "Predicting Predictability," Preprint, Department of Computer Science, University of Colorado at Boulder, in preparation, 1994.

[10] A.M. Fraser and A. Dimitriadis, "Forecasting Probability Densities Using Hidden Markov Models with Mixed States." In *Time Series Prediction: Forecasting the Future and Understanding the Past*, A.S. Weigend and N.A. Gershenfeld, eds., Addison-Wesley, pp. 265–282, 1994.